pyexsim12

This notebook is prepared to illustrate the use of pyexsim12, a Python module to run the stochastic ground motion simulation code EXSIM12 and visualize and postprocess the results.

First, we import pyexsim12, pyplot, pandas and numpy

In [1]:
```python
from pyexsim12 import *
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

In order to run a ground motion simulation with pyexsim12, we first need to create several objects for our simulation case, representing different components. These objects are:

- Source()
- Path()
- Amplification()
- Misc()
- Sites()

First we will create the Source object. A Source object requires 4 inputs for initialization, which can be seen by calling the help function on Source class as: `help(Source)`. These 4 inputs are:

- source_spec - SourceSpec(mw, stress_drop, kappa, kappa_flag)
- fault_geom - FaultGeom(fault_edge, angles, fault_type, len_width)
- hypo - Hypocenter(hypo_along_fault, hypo_down_dip)
- rupture - Rupture(vrup_beta, slip_weights)

In the cell below, we create each of these 4 input objects along with a custom slip matrix. We save this custom slip matrix with the `simulation.create_slip_file(slip_matrix, slip_weights)` method. We then pass this slip file into the Rupture object. Finally, we create the Source object with all the four inputs, and print the object to see key information.

In [2]:
```python
# %% Creating the Source object
src_spec = SourceSpec(mw=7.1, stress_drop=100, kappa=0.047)  # kappa_flag = 1 by default, so no need to change
fault_geom = FaultGeom(fault_edge=(40.77, 31.45),
                       angles=[264.0, 64.0, 0.0],
                       fault_type="S",
                       len_width=[65.0, 25.0, 5.0, 5.0, 70.0])
hypo = Hypocenter(hypo_along_fault=7, hypo_down_dip=3)
# Create the slip file
# First we will create a matrix of slip weights. Since we have 65/5=13 subfaults along the length, and 25/5=5 a
# width, dimensions of the array will be 5 x 13.
slip_matrix = np.array([[0.4, 0.6, 1.2, 1.35, 1.05, 1.2, 0.8, 1.2, 1.8, 1.7, 1.3, 0.9, 0.6],
                        [0.4, 0.9, 1.3, 1.4, 2.0, 1.8, 1.1, 1.8, 2.7, 2.6, 1.9, 1.3, 0.75],
                        [0.28, 0.55, 0.9, 1.2, 1.5, 1.7, 1.2, 2.1, 2.6, 2.4, 1.8, 1.2, 0.75],
                        [0.1, 0.25, 0.6, 0.7, 1.05, 1.35, 1.5, 1.95, 2.1, 1.6, 1.05, 0.75, 0.6],
                        [0.1, 0.1, 0.3, 0.45, 0.5, 0.7, 1.0, 1.2, 1.2, 0.6, 0.3, 0.2, 0.1]])
# Now we will pass this array into the create_slip_file method of simulation module. Since exsim is located at
# folder "exsim12", which is the default argument for the exsim_folder parameter, we will leave it as default.
simulation.create_slip_file(slip_matrix=slip_matrix, filename="slip_weights.txt")
rupture = Rupture(vrup_beta=0.8, slip_weights="slip_weights.txt")
src = Source(src_spec, fault_geom, hypo, rupture)

# Print the recently created src object to see basic information:
print(src)
```

```
Mw: 7.1
Stress drop: 100 bars
kappa: 0.047 s
Strike: 264.0°
Dip: 64.0°
Depth: 0.0 km
```

In this cell, we create the Path object with 5 inputs:

- time_pads - TimePads(tpad1, tpad2, delta_t)
- crust - Crust(beta, rho)
- geometric_spreading - GeometricSpreading(n_seg, spread)
- quality_factor - QualityFactor(q_min, q_zero, eta)
- path_duration - PathDuration(n_dur, r_dur, dur_slope)

In [3]:
```python
# %% Creating the Path object

time_pads = TimePads(tpad1=50.0, tpad2=20.0, delta_t=0.005)
crust = Crust(beta=3.7, rho=2.8)
geom_spread = GeometricSpreading(n_seg=3, spread=[(1.0, -1.0), (30.0, -0.6), (50.0, -0.5)])
q_factor = QualityFactor(q_min=0.0, q_zero=88, eta=0.9)
path_dur = PathDuration()  # No input is provided as the default values will be used
path = Path(time_pads, crust, geom_spread, q_factor, path_dur)
```

The Amplification object only requires three inputs, filenames of site, crustal and empirical amplification files. These files can created with `simulation.create_amp` method, or they can be manually prepared, in which case the corresponding filename should be passed as an input argument to the Amplification object. Here, we create custom amplification files for site and crustal amplification. We do not pass in an argument for the empirical amplification file, which will have the default value of `empirical_amps.txt`, which is the file distributed with EXSIM12 with no empirical amplification.

In [4]:
```python
# %% Creating amplification files and Amplification object
simulation.create_amp(freq=[0.1953, 0.9766, 5.859, 8.887, 11.72],
                      amp=[5.115, 7.155, 0.7477, 0.5308, 0.2812],
                      filename="site_amp_tutorial.txt",
                      header="Site amplification file for pyexsim12 tutorial")

simulation.create_amp(freq=[0.01, 0.1, 0.2, 0.3, 0.5, 0.9, 1.25, 1.8, 3.0, 5.3, 8.0, 14.0],
                      amp=[1.0, 1.02, 1.03, 1.05, 1.07, 1.09, 1.11, 1.12, 1.13, 1.14, 1.15, 1.15],
                      filename="crustal_amp_tutorial.txt",
                      header="Crustal amplification file for pyexsim12 tutorial")

amp = Amplification(site_amp="site_amp_tutorial.txt", crustal_amp="crustal_amp_tutorial.txt")
```

Finally, we create the Misc and Sites objects. We do not provide any arguments for the Misc object and keep everything as default. And we provide the coordinates for only one site for the Sites object.

In [5]:
```python
# %% Create Misc and Sites objects

misc = Misc()  # Everything will be kept as default
sites = Sites([(40.85, 31.17)])  # Only one site will be used for demonstration
```

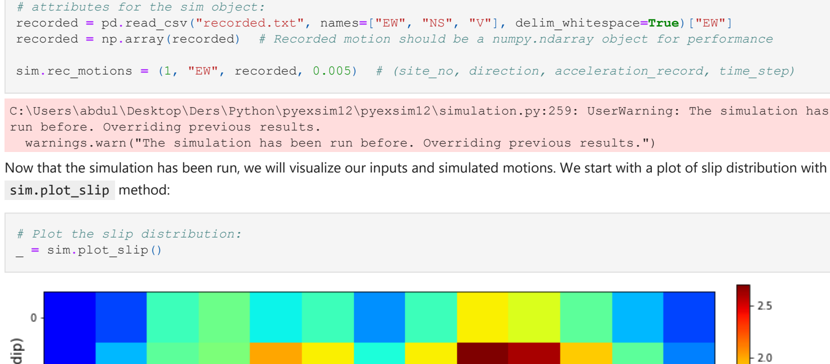
Now the Simulation object is ready for initialization. After creating the Simulation object with the name of "sim", we create the input file with `sim.create_input_file` method. We then run the simulation with `sim.run` method, with the override argument equal to True, which will override the results if the simulation for the same inputs has been run before. This overriding will also raise a warning as shown in the output of the cell below. After running the simulation, we read the recorded acceleration file for the simulation, and store this record in our sim object as the `sim.rec_motions` attribute for the "EW" direction.

In [6]:
```python
# %% Now the Simulation object is ready for initialization.
sim = Simulation(src, path, amp, misc, sites)
sim.create_input_file(save=True)  # Create the input file for EXSIM12
sim.run(override=True)  # Run the simulation, if output files for this configuration exist, they will be overwi

# Recorded acceleration series for this simulation exist in the current working directory. They will be passed
# attributes for the sim object:
recorded = pd.read_csv("recorded.txt", names=["EW", "NS", "V"], delim_whitespace=True)["EW"]
recorded = np.array(recorded)  # Recorded motion should be a numpy.ndarray object for performance

sim.rec_motions = (1, "EW", recorded, 0.005)  # (site_no, direction, acceleration_record, time_step)
```

```
C:\Users\abdul\Desktop\Ders\Python\pyexsim12\pyexsim12\simulation.py:259: UserWarning: The simulation has been
run before. Overriding previous results.
  warnings.warn("The simulation has been run before. Overriding previous results.")
```

Now that the simulation has been run, we will visualize our inputs and simulated motions. We start with a plot of slip distribution with the `sim.plot_slip` method:

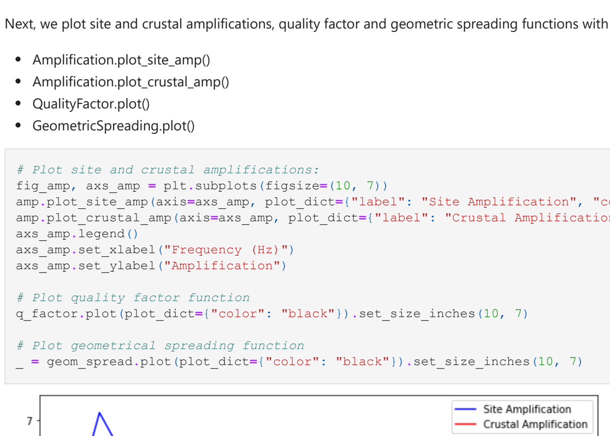
In [7]:
```python
# Plot the slip distribution:
_ = sim.plot_slip()
```

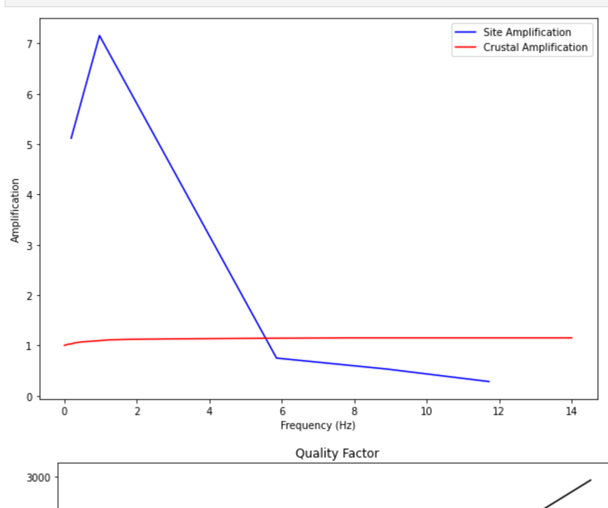


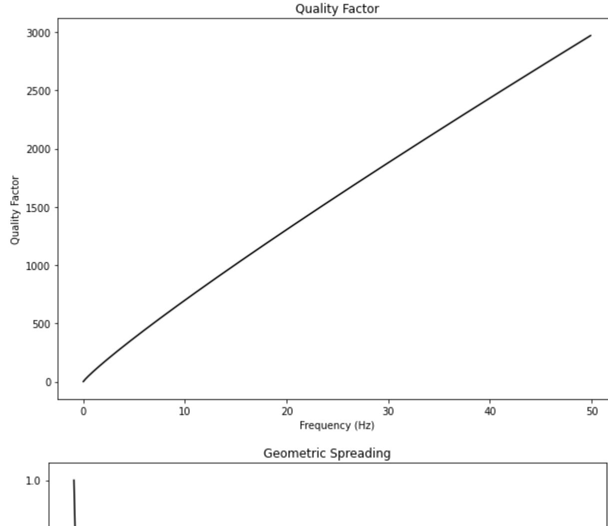
Next, we plot site and crustal amplifications, quality factor and geometric spreading functions with the methods:

- Amplification.plot_site_amp()
- Amplification.plot_crustal_amp()
- QualityFactor.plot()
- GeometricSpreading.plot()

In [8]:
```python
# Plot site and crustal amplifications:
fig_amp, axs_amp = plt.subplots(figsize=(10, 7))
amp.plot_site_amp(axis=axs_amp, plot_dict={"label": "Site Amplification", "color": "blue"})
amp.plot_crustal_amp(axis=axs_amp, plot_dict={"label": "Crustal Amplification", "color": "red"})
axs_amp.legend()
axs_amp.set_xlabel("Frequency (Hz)")
axs_amp.set_ylabel("Amplification")

# Plot quality factor function
q_factor.plot(plot_dict={"color": "black"}).set_size_inches(10, 7)

# Plot geometrical spreading function
_ = geom_spread.plot(plot_dict={"color": "black"}).set_size_inches(10, 7)
```

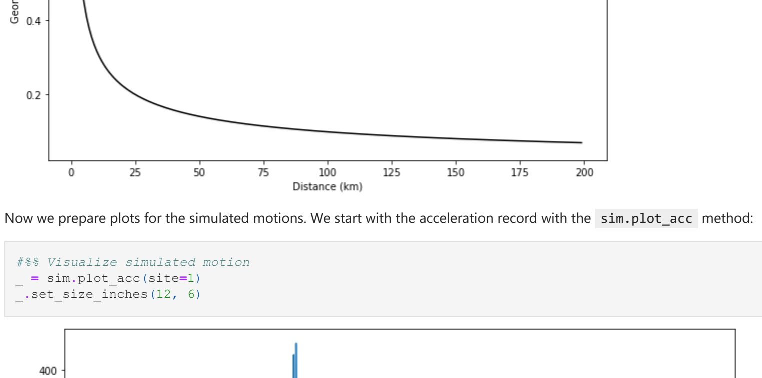






Now we prepare plots for the simulated motions. We start with the acceleration record with the `sim.plot_acc` method:

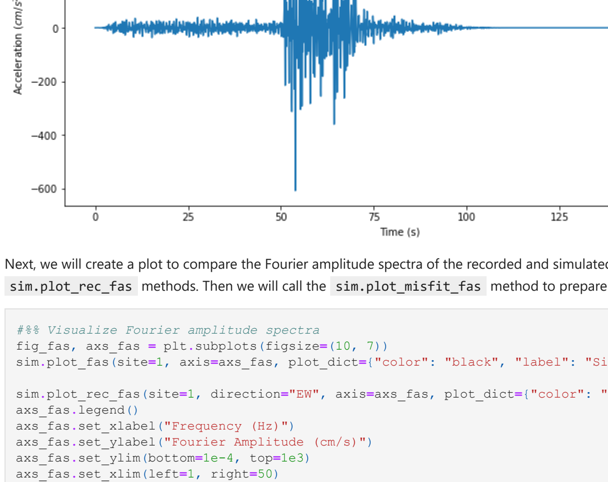
In [9]:
```python
#%% Visualize simulated motion
_ = sim.plot_acc(site=1)
_.set_size_inches(12, 6)
```

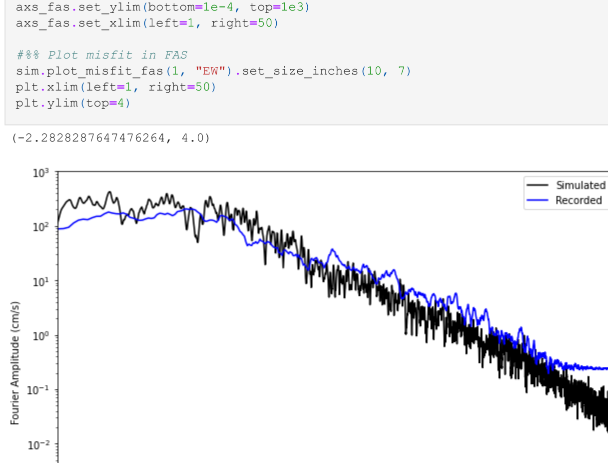


Next, we will create a plot to compare the Fourier amplitude spectra of the recorded and simulated motions, with `sim.plot_fas` and `sim.plot_rec_fas` methods. Then we will call the `sim.plot_misfit_fas` method to prepare a plot for the misfit in terms of FAS.

In [10]:
```python
#%% Visualize Fourier amplitude spectra
fig_fas, axs_fas = plt.subplots(figsize=(10, 7))
sim.plot_fas(site=1, axis=axs_fas, plot_dict={"color": "black", "label": "Simulated"})

sim.plot_rec_fas(site=1, direction="EW", axis=axs_fas, plot_dict={"color": "blue", "label": "Recorded"})
axs_fas.legend()
axs_fas.set_xlabel("Frequency (Hz)")
axs_fas.set_ylabel("Fourier Amplitude (cm/s)")
axs_fas.set_ylim(bottom=1e-4, top=1e3)
axs_fas.set_xlim(left=1, right=50)

#%% Plot misfit in FAS
sim.plot_misfit_fas(1, "EW").set_size_inches(10, 7)
plt.xlim(left=1, right=50)
plt.ylim(top=4)
```

Out[10]:
```
(-2.2828287647476264, 4.0)
```



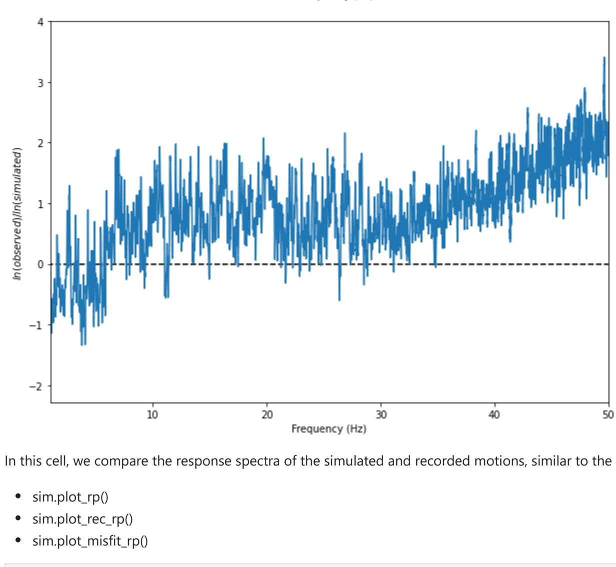


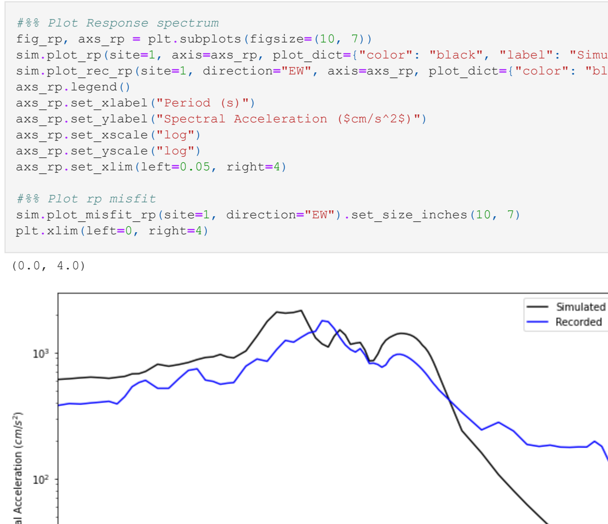
In this cell, we compare the response spectra of the simulated and recorded motions, similar to the FAS, using the following methods:

- sim.plot_rp()
- sim.plot_rec_rp()
- sim.plot_misfit_rp()

In [11]:
```python
#%% Plot Response spectrum
fig_rp, axs_rp = plt.subplots(figsize=(10, 7))
sim.plot_rp(site=1, axis=axs_rp, plot_dict={"color": "black", "label": "Simulated"})
sim.plot_rec_rp(site=1, direction="EW", axis=axs_rp, plot_dict={"color": "blue", "label": "Recorded"})
axs_rp.legend()
axs_rp.set_xlabel("Period (s)")
axs_rp.set_ylabel("Spectral Acceleration ($cm/s^2$)")
axs_rp.set_xscale("log")
axs_rp.set_yscale("log")
axs_rp.set_xlim(left=0.05, right=4)

#%% Plot rp misfit
sim.plot_misfit_rp(site=1, direction="EW").set_size_inches(10, 7)
plt.xlim(left=0, right=4)
```

Out[11]:
```
(0.0, 4.0)
```





Finally, we plot the simulated motion in comparison with the ground motion model of Boore et al. (2014). We also prepare a plot of normalized residuals calculated from the model. We use following methods to accomplish these:

- sim.plot_rp()
- sim.plot_bssa14()
- sim.plot_bssa14_eps()

In [12]:
```python
#%% Plot GMM
fig_gmm, axs_gmm = plt.subplots(figsize=(10, 7))
sim.plot_rp(site=1, axis=axs_gmm, plot_dict={"color": "black", "label": "Simulated"})
sim.plot_bssa14(site=1, vs30=300, axis=axs_gmm, plot_dict={"color": "blue", "label": "BSSA14"})

axs_gmm.legend()
axs_gmm.set_xlabel("Period (s)")
axs_gmm.set_ylabel("Spectral Acceleration ($cm/s^2$)")
axs_gmm.set_xscale("log")
axs_gmm.set_yscale("log")
axs_gmm.set_xlim(left=0.1)

# Plot GMM epsilons
sim.plot_bssa14_eps(site=1, vs30=300, plot_dict={"color": "black"}).set_size_inches(10, 7)
plt.xscale("log")
plt.xlim(left=0.1)
```

Out[12]:
```
(0.1, 14.12537544622754)
```